

01 Setting up and Basic commands:

Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.

Solution:-

To initialize a new Git repository in a directory, create new file, add it to the staging area and commit the changes with an appropriate commit message, follow these steps:

1) Open your terminal and navigate to the directory where you want to create the Git repository.

2) Initialize a new Git repository in that directory.

[\$ git init]

3) Create a new file in the directory. For example, let's create a file named "my-file.txt". You can use any text editor @ command-line tools to create the file.

4) Add the newly created file to the staging area. Replace "my-file.txt" with the actual name of your file:

Acharya

Teacher's Signature

[+] gift add my-file.txt
How's command along the file for the upcoming

5) command the changes will be done on other people's computer
message. Purpose "your command message here" will be
received through the output of your change.

[+] gift commit in "your command message here" []
your command messages should easily decide the purpose
of output of the change you made

[+] gift commit - in "add now the my-file.txt
for eg:-
git add my-file.txt
git commit -m "My First Commit"
git push [root-commit] 390f4d] My First Commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 my-file.txt

After these steps, your changes will be committed to the new
git space to which the provided command belongs.
now have a relation of the space with the new
and if the new should be git.

```
MINGW64 /c/Users/848/Desktop/GIT-LAB
$ git config --global user.name "Buddha Venkatesh"
$ git config --global user.email "buddha.23.becsa@charly.in"
$ init initialized empty Git repository in C:/Users/848/Desktop/GIT-LAB/
$ touch my-file.txt
$ git add my-file.txt
$ git commit -m "My First Commit"
$ git push [root-commit] 390f4d] My First Commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 my-file.txt
$ git add my-file.txt
$ git commit -m "My First Commit"
$ git push [root-commit] 390f4d] My First Commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 my-file.txt
$ !
```

```
MINGW64 /c/Users/848/Desktop/GIT_LAB
$ git add .
$ git commit -m "My Second Checkout"
[feature-branch 6a62de1] My Second Checkout
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 my_file2.txt

MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)
$ git checkout master
Switched to branch 'master'

MINGW64 /c/Users/848/Desktop/GIT_LAB (master)
$ git merge feature-branch
Updating 3910f4d..6a62de1
Fast-forward
  my_file2.txt | 0
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 my_file2.txt

MINGW64 /c/Users/848/Desktop/GIT_LAB (master)
```

Q2 Creating and managing Branches:-

Create a new branch named "feature-branch", switch to the "master" branch, merge the "feature-branch" into "master".

Solution:-

To create a new branch named "feature-branch", switch to the "master" branch, and merge the "feature-branch" into "master" in git, follow these steps:

1) make sure you are in the "master" branch by switching to it:

[\$ git checkout master]

2) Create a new branch named "feature-branch" and switch to it

[\$ git checkout -b feature-branch]

This command will create a new branch called "feature-branch" and switch to it.

3) make your changes in the "feature-branch" by adding, modifying or deleting file as needed

4) Stage and commit your changes in the "feature-branch":

[\$ git add.]

[\$ git commit -m "your commit message for feature-branch"]

Replace "your commit message for feature-branch" with a descriptive commit message for the changes you made in the "feature-branch".

5) Switch back to the "master" branch.

[\$ git checkout master]

6) merge the "feature-branch" into "master" branch:

[\$ git merge feature-branch]

This command will incorporate the changes from the "feature-branch" into "master" branch.

Now, your changes from the "feature-branch" have been merged into the "master" branch, your project's history will reflect the changes made in both branches.

03 Creating and managing Branches:

Write the command to stash your changes, switch branches, and then apply the stashed changes.

Solution:-

To stash your changes, switch branch, and then apply the stashed changes in it, you can use the following commands.

1) Stash your changes:

[`$ git stash save "your stash message"`]

This command will save your changes in a stash, which acts like a temporary storage for changes that are not ready to be committed.

2) Switch to the desired branch:

[`$ git checkout target-branch`]

Replace "target-branch" with the name of the branch you want to switch to

3) Apply the stashed changes:

```
warning: in the working copy of 'my_file2.txt', LF will be replaced by CRLF the next time Git touches it
Saved working directory and index state. On master: My First Stash
```

```
348@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (master)
```

```
$ git checkout feature-branch
Switched to branch 'feature-branch'.
```

```
348@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)
$
```

```
348@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)
$ git stash apply
On branch feature-branch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   my_file2.txt
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

```
348@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)
$ git add .
$
```

```
348@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)
$ git stash apply
On branch feature-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   my_file2.txt
```

```
348@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)
$
```

[`git stash apply`]

This command will apply the most recent stash to your current working branch. If you have multiple stashes, you can specify a stash by name or reference (e.g: `git stash apply stash@{2}`) if needed.

If you want to remove the stash after applying it, you can use `git stash pop` instead of `git stash apply`.

Remember to replace "your stash message" and "target-branch" with the actual message you want for your stash and the name of the branch you want to switch to.

MINGW64:/c/Users/Bharathi P/OneDrive/Desktop/GIT

Bharathi PAGE SKTOP-2N84T20 MINGW64 ~/OneDrive/Desktop/GIT

\$

Bharathi PAGE SKTOP-2N84T20 MINGW64 ~/OneDrive/Desktop/GIT

\$

Bharathi PAGE SKTOP-2N84T20 MINGW64 ~/OneDrive/Desktop/GIT

\$

```
Sharathi PAGE SKTOP-2N84T20 MINGW64 ~/OneDrive/Desktop/GIT
$ git clone https://github.com/BuddhaLa-Venkatesh/MyAPP.git
Cloning into 'MyAPP' ...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3) done.
remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.
```

Bharathi PAGE SKTOP-2N84T20 MINGW64 ~/OneDrive/Desktop/GIT

\$ |

Experiment No.	DL	Date	Page No.
Q1) *Collaboration and Remote Repositories:- *			
clone a remote git repository to your local machine.			
Solution:-			
To clone a remote git repository to your local machine , follow these steps:-			
<p>1) Open your terminal @ command prompt .</p> <p>2) Navigate to the directory where you want to the clone the remote git repository . You can see the cd command to change your working directory .</p> <p>3) Use the git clone command to clone the remote repository . Replace <repository_url> with the URL of the remote git repository you want to clone .</p>			
<p>For example:- If you were cloning a repository from GitHub , the URL might look like this :</p>			
<pre>[\$ git clone <repository_url>]</pre>			
<p>We'll see full example .</p>			
<pre>[\$ git clone https://github.com/username/repo-name.git]</pre>			
Teacher's Signature _____			
Acharya			

Replace `https://github.com/username/repo-name.git` with the actual URL of the repository you want to clone.

My git will clone the repository to your local machine. Once the process is complete, you will have a local copy of the remote repository in your chosen directory.

You can now work with the cloned repository on your local machine, make changes, and push those changes back to the remote repository as needed.

Teacher's Signature _____

Acharya

MINIGW64\q\Users\bharathi P\OneDrive\Desktop\GIT\MyAPP

```
bharathi @DESKTOP-2N84T20 MINGW64 ~ /OneDrive/Desktop/GIT/MyAPP
$ git clone https://github.com/Buddda1a-Venkatesh/MyAPP.git
Cloning into 'MyAPP'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 12 (delta 0), reused 6 (delta 0), pack-reused 0
Receiving objects: 100% (12/12), done.

Bharathi @DESKTOP-2N84T20 MINGW64 ~ /OneDrive/Desktop/GIT/MyAPP
$ cd MyAPP/
Bharathi @DESKTOP-2N84T20 MINGW64 ~ /OneDrive/Desktop/GIT/MyAPP (main)
$ git checkout -b new-branch
Switched to a new branch 'new-branch'.

Bharathi @DESKTOP-2N84T20 MINGW64 ~ /OneDrive/Desktop/GIT/MyAPP (main)
$ git fetch origin
Bharathi @DESKTOP-2N84T20 MINGW64 ~ /OneDrive/Desktop/GIT/MyAPP (new-branch)
$ git rebase origin
Current branch new-branch is up to date.

Bharathi @DESKTOP-2N84T20 MINGW64 ~ /OneDrive/Desktop/GIT/MyAPP (new-branch)
$ touch new_file.txt
Bharathi @DESKTOP-2N84T20 MINGW64 ~ /OneDrive/Desktop/GIT/MyAPP (new-branch)
$ git add .
bash: git: command not found
Bharathi @DESKTOP-2N84T20 MINGW64 ~ /OneDrive/Desktop/GIT/MyAPP (new-branch)
$ git add .
Bharathi @DESKTOP-2N84T20 MINGW64 ~ /OneDrive/Desktop/GIT/MyAPP (new-branch)
$ git push origin new-branch
Everything up-to-date

Bharathi @DESKTOP-2N84T20 MINGW64 ~ /OneDrive/Desktop/GIT/MyAPP (new-branch)
$
```

05 * Collaboration and Remote Repositories

Fetch the latest changes from a remote repository & release your local branch onto the updated remote branch.

Solution:-

To get the latest changes from a remote repository -
say and release your local branch onto the updated
remote branch in Git, follow these steps:

1) Open your terminal & command prompt.

git

2) make sure you are in the local branch that you want to release. You can switch to the branch using the following command, replacing ~~branch-name~~ with your actual branch name:

[\$ git checkout <branch-name>]

3) Fetch the latest changes from the remote repository. This will update your local repository with the changes from the remote without merging them into your local branch.

Teacher's Signature _____

Acharya

[\$ git fetch origin]

Here, origin is the default name for the remote repository. If you have multiple remotes, replace origin with the name of the specific remote you want to fetch from.

4) Once you have fetched the latest changes, release your local branch onto the updated remote branch:

[\$ git release origin / 2branch-name]

Replace 2branch-name with the name of the remote branch you want to release onto. This command will ~~reapply~~ your local working on top of the latest changes from the remote branch, effectively incorporating the remote changes into your branch history.

Synchronize any conflicts that may arise during the release process. Git will skip and notify you if there are conflicts that need to be resolved. Use a text editor to edit the conflicting files, save the changes, and the continue the release with:

[\$git rebore --continue]

By applying resolving any conflicts and completing the rebore, you have successfully updated your local branch with the latest changes from the remote branch.

If you want to push your released changes to the remote repository, use the git push command. However, be cautious when pushing to a shared remote branch, as it can potentially overwrite other developer's changes.

 [\$git push origin <branch-name>]

Replace <branch-name> with the name of your local branch. By following these steps, you can keep your local branch up to date with the latest changes from the remote repository and maintain a clean and linear history through rebasing.

MINGW64:/c/Users/848/Desktop/GIT_LAB

848@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)

\$ git checkout master

M my_file2.txt

Switched to branch 'master'

848@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (master)

\$

848@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (master)

\$ git merge feature-branch -m "My First Merge"

Already up to date.

848@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (master)

\$ |

6



Scanned with OKEN Scanner

Q6

* Collaboration and Remote Repositories: *

write the command -is merge "feature-branch" into "master" while providing a custom commit message for the merge.

Solution:-

To merge the "feature-branch" into "master" in Git while providing a custom commit message for the merge, you can use the following command:

```
$ git checkout master
```

```
$ git merge feature-branch -m "your custom commit here"
```

Replace "your custom commit message here" with a meaningful and descriptive commit message for the merge. This message will be associated with the merge commit that is created when you merge "feature-branch" into "master".

MINGW64:/c/Users/848/Desktop/GIT_LAB

27q

```
$ git log --author="Buddala Venkatesh" --since="2024-04-01" --until="2024-04-16"
commit 6a62de1ea10e313495c543e90fe551952bd1d1c (HEAD -> master, tag: v1.0, feature-branch)
Author: Buddala Venkatesh <buddala.23.becs@acharya.in>
Date:   Tue Apr 16 13:19:34 2024 +0530
```

My Second Checkout

```
commit 3910f4d7ee63264f922eaa41d28ab2b84fb8290f (tag: v2.0)
Author: Buddala Venkatesh <buddala.23.becs@acharya.in>
Date:   Mon Apr 15 19:04:37 2024 +0530
```

My First Commit

```
348AICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (master)
$
```



Scanned with OKEN Scanner

Q2

* Git Tags and Releases *

Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.

Solution:

To create a lightweight Git tag name "v1.0" for a commit in your local repository, you can use the following command:

[\$ git tag v1.0]

This command will create a lightweight tag called "v1.0" for the most recent commit in your current branch. If you want to tag a specific commit other than the most recent one, you can specify the commit's SHA-1 hash after the tag name.

For example:

[\$ git tag v1.0 <commit-SHA>]

Replace <commit-SHA> with the actual SHA-1 hash of the commit you want to tag.

08 *Advanced Git Operations: *

Write the command to cherry-pick range of commits from "source-branch" to the current branch.

Solution:

To cherry-pick a range of commits from "source-branch" to the current branch, you can use the following command:

[\$ git cherry-pick <start-commit>..<end-commit>]

Replace <start-commit> with the commit at the beginning of the range, and <end-commit> with the commit at the end of the range. The symbol . is used to exclude the <start-commit> itself & include all commits after it up to and including <end-commit>. This will apply the changes from the specified range of commits to your current branch.

For example, if you want to cherry-pick a range of commits from "source-branch" starting from commit ABC123 and ending at commit DGF456, you would use:

Teacher's Signature _____

Acharya

```
B
sharathip@DESKTOP-2N84T20 MINGW64 ~/OneDrive/Desktop/GIT/trimstray
$ git clone https://github.com/trimstray/trimstray.git
Cloning into 'trimstray'...
remote: Enumerating objects: 109, done.
remote: Counting objects: 100% (109/109), done.
remote: Compressing objects: 100% (73/73), done.
remote: Total 109 (delta 33), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (109/109), 24.02 KiB | 6.00 MiB/s, done.
Resolving deltas: 100% (33/33), done.

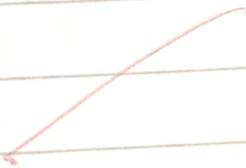
sharathip@DESKTOP-2N84T20 MINGW64 ~/OneDrive/Desktop/GIT
$ cd trimstray/
```

```
sharathip@DESKTOP-2N84T20 MINGW64 ~/OneDrive/Desktop/GIT/trimstray (main)
$
```

```
sharathip@DESKTOP-2N84T20 MINGW64 ~/OneDrive/Desktop/GIT/trimstray (main)
$ git cherry-pick 0de189a..fa374
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
error: could not apply 0de189e..fa374
hint: After resolving the conflicts, mark them with
hint: "git add/rm <pathspec>" ; then run
hint: "git cherry-pick --continue".
hint: You can instead skip this commit with "git cherry-pick --skip".
hint: To abort and get back to the state before "git cherry-pick",
hint: run "git cherry-pick --abort".
sharathip@DESKTOP-2N84T20 MINGW64 ~/OneDrive/Desktop/GIT/trimstray (main|CHERRY_PICKING)
$
```

[\$git cherry-pick ABC123^.. DEF456]

make sure you are on the branch where you
want to apply these changes before running the
cherry-pick command.



*Analysing and changing Git History *

Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date and commit message?

Solution:

To view the details of a specific commit, including the author, date and commit message, you can use the `git show` or `git log` command with the commit ID. Here are both options:

1) using `git show`

bash

```
git show <commit-ID>
```

Replace `<commit-ID>` with the actual commit ID you want to view. This command will display detailed information about the specified commit, including the commit message, author, date and the changes introduced by that commit.

For example:-

```
$git show abc123
```

MINGW64:/c/Users/848/Desktop/GIT_LAB 9

```
$ MINGW64: /c/Users/848/Desktop/GIT_LAB (master)
$ git log -n 5
commit 6a62de11ea10e313495c543e90fe551952bd1d1c (HEAD -> master, tag: v1.0, feature-branch)
Author: Buddala Venkatesh <buddala.23.becs@acharya.in>
Date:  Tue Apr 16 13:19:34 2024 +0530
```

My Second Checkout

```
commit 3910f4d7ee63264f922eaa41d28ab2b84fb8290f (tag: v2.0)
Author: Buddala Venkatesh <buddala.23.becs@acharya.in>
Date:  Mon Apr 15 19:04:37 2024 +0530
```

My First Commit

```
48@ICT-848 MINGW64: /c/Users/848/Desktop/GIT_LAB (master)
```



By using `git log`:

```
$ git log -n 1 <commit-ID>
```

The `-n 1` option tells Git to show only one commit. Replace `<commit-ID>` with the actual commit ID. This command will display a condensed view of the specified commit, including its commit message, author, date and commit ID.

For example:

```
$ git log -n 1 abc123
```

Both of these commands will provide you with the necessary information about the specific commit you're interested in.

```
MINGW64/c/Users/848/Desktop/GIT_LAB /o
$ git show
commit 6a62de11ea10e313495c543e90fe551952bd1d1c (HEAD -> master, tag: v1.0, feature-branch)
Author: Buddala Venkatesh <buddala.23.becs@acharya.in>
Date:  Tue Apr 16 13:19:34 2024 +0530

My Second Checkout

diff --git a/my_file2.txt b/my_file2.txt
new file mode 100644
index 0000000..e69de2d

848@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (master)

$ git log -n 1
commit 6a62de11ea10e313495c543e90fe551952bd1d1c (HEAD -> master, tag: v1.0, feature-branch)
Author: Buddala Venkatesh <buddala.23.becs@acharya.in>
Date:  Tue Apr 16 13:19:34 2024 +0530

My Second Checkout

$
```

10 * Analysing and changing Git History *

write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31".

Solution:

To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31" in Git, you can use the `git log` command with the `--author` and `--since` and `--until` options. Here's the command:

\$ git log
\$ git log --author = "JohnDoe" --since = "2023-01-01" --
until = "2023-12-31"

This command will display a list of commits made by the author "JohnDoe" that fall within the specified date range, from January 1, 2023 to December 31, 2023. Make sure to adjust the author name and date range as needed for your specific use case.

MINGW64/c/Users/848/Desktop/GIT_LAB

Date: Tue Apr 16 13:19:34 2024 +0530

My Second Checkout

```
commit 3910f4d7ee63264f922eaa41d28ab2b84fb8290f
Author: Buddala Venkatesh <buddala.23.becs@acharya.in>
Date: Mon Apr 15 19:04:37 2024 +0530
```

My First Commit

```
848@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)
$ git tag v1.0 3910f4d7ee63264f922eaa41d28ab2b84fb8290f
fatal: tag 'v1.0' already exists

848@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)
$ git tag v2.0 3910f4d7ee63264f922eaa41d28ab2b84fb8290f
848@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)
$ git log
commit 6a62de11ea10e313495c543e90fe551952bd1d1c (HEAD -> feature-branch, tag: v1.0, master)
Author: Buddala Venkatesh <buddala.23.becs@acharya.in>
Date: Tue Apr 16 13:19:34 2024 +0530
```

My Second Checkout

```
commit 3910f4d7ee63264f922eaa41d28ab2b84fb8290f (tag: v2.0)
Author: Buddala Venkatesh <buddala.23.becs@acharya.in>
Date: Mon Apr 15 19:04:37 2024 +0530
```

My First Commit

```
848@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (feature-branch)
```

11 * Analysing and changing Git History *

write the command to display the last five commits in the repository's history.

Solution:

To display the last five commits in a Git repository's history, you can use the `git log` command with the `-n` option, which limits the number of displayed commits. Here's the command:

[`$ git log -n 5`]

This command will show the last five commits in the repository's history. You can adjust the number after `-n` to display a different number of commits if needed.

```
MINGW64:/c/Users/848/Desktop/GIT_LAB
```

```
Reapply "My First Commit-Reverted"
```

```
This reverts commit 1efe7d6aa28893b5a94078fela342fae0db21a33.
```

```
Please enter the commit message for your changes. Lines starting  
with '#' will be ignored, and an empty message aborts the commit.
```

```
On branch master
```

```
You are currently reverting commit 6a62de1.
```

```
Changes to be committed:
```

```
new file: my_file.txt
```

```
MINGW64:/c/Users/848/Desktop/GIT_LAB
```

```
[master 6563074] Reapply "My First Commit-Reverted-2nd Time"  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 my_file.txt
```

```
848@ICT-848 MINGW64 /c/Users/848/Desktop/GIT_LAB (master)
```

* Analyzing and changing git history *

While the command to undo the changes introduced by the commit with the ID "abc123"

Solution:

To undo the changes introduced by a specific commit with the ID "abc123" in Git, you can use the `git revert` command. The `git revert` command creates a new commit that undoes the changes made by the specified commit, effectively "reverting" the commit here's the command.

[\$ git revert abc123]

~~Replace "abc123"~~ with the actual commit ID that you want to revert. After running this command, Git will create a new commit that negates the changes introduced by the specified commit. This is a safe way to undo changes in Git because it preserves the commit history and creates a new command to record the reversal of the changes.